Dijkstra

Dijkstra's algorithm is a shortest-path algorithm used to find the minimum distance from a starting node to all other nodes in a weighted graph where all edge weights are non-negative. It works by gradually exploring the graph, always choosing the unvisited node with the smallest known distance from the start. Initially, the starting node is assigned distance 0 and all others infinity. The algorithm repeatedly picks the closest unvisited node, updates the distances to its neighbors if a shorter path is found, and marks it as visited. This continues until all reachable nodes have been processed. Dijkstra's algorithm is widely used in routing, GPS navigation, and network optimization because it efficiently provides the shortest path in weighted graphs. It typically uses a priority queue to speed up the process, giving it a time complexity of O((V + E) log V) when implemented with a min-heap.

Advantages

- Finds shortest path in weighted graphs (as long as weights are non-negative).
- Efficient with priority queue → suitable for large graphs.
- Produces shortest distance to all nodes simultaneously (single-source shortest path).
- Useful in network routing, GPS navigation, map applications, pathfinding.

Limitations

- and Cannot handle graphs with negative weights (fails or gives wrong results).
- Requires additional data structures (priority queue), making it more complex than BFS.
- May be slow on very dense graphs without optimization.
- Does not work efficiently if all nodes must be revisited many times (e.g., dynamic weights).

Steps

Initial Setup

1. Set dist(1)=0, dist(2)=∞, dist(3)=∞, dist(4)=∞, dist(5)=∞.
2. Insert (1,0) into priority queue

⭐ Detailed Steps (Each Line Has Node Processed + Updates)

3. Pick node 1 (distance 0) → visit it.

4. From 1 to 3: new dist = 0+2 = 2 → update dist(3)=2.
5. From 1 to 4: new dist = 0+2 = 2 → update dist(4)=2.
6. From 1 to 2: new dist = 0+4 = 4 → update dist(2)=4.
7. From 1 to 5: new dist = 0+5 = 5 → update dist(5)=5.
   (Queue now: 3(2), 4(2), 2(4), 5(5)
8. Pick node 3 (distance 2) → visit it.
9. From 3 to 1 → already visited → skip.
10. From 3 to 4: new dist = 2+3 = 5 → current dist(4)=2 → no update
11. Pick node 4 (distance 2) → visit it.
12. From 4 to 1 → visited → skip.
13. From 4 to 3 → visited → skip.
14. From 4 to 5: new dist = 2+2 = 4 → current dist(5)=5 → update dist(5)=4.
    (Queue now: 2(4), 5(4)
15. Pick node 2 (distance 4) → visit it.
16. From 2 to 1 → visited → skip.
17. From 2 to 5: new dist = 4+1 = 5 → current dist(5)=4 → no update.
18. Pick node 5 (distance 4) → visit it.
19. From 5 to 1 → visited → skip.
20. From 5 to 2 → visited → skip.
21. From 5 to 4 → visited → skip

⭐ Final Distances from Node 1

dist(1) = 0

dist(3) = 2

dist(4) = 2

dist(2) = 4

dist(5) = 4

Pseudocode (Simple Standard Version)

```
Dijkstra(graph, start):

    for each node in graph:
        distance[node] = infinity
    distance[start] = 0

    create min_priority_queue Q
    Q.push(start, 0)

    while Q is not empty:
        current = Q.pop_min()

        for each (neighbor, weight) of current:
            new_dist = distance[current] + weight

            if new_dist < distance[neighbor]:
                distance[neighbor] = new_dist
                Q.push(neighbor, new_dist)

    return distance
```

Time Complexity

Depends on data structure used for selecting the minimum distance node:

Implementation	Data Structure	Time Complexity
Simple array	Array	$O(V^2)$
Efficient	Min-heap (priority queue)	$O((V + E) \log V)$
Most efficient	Fibonacci heap	$O(E + V \log V)$

Where
V = number of vertices
E = number of edges

Dijkstra Code:

https://github.com/pulokdas062/GraphAlgorithm/blob/main/Dijkstra/Dijkstra%20Basic/dijkstra.cpp